**Socker: Δίκτυο-εφαρμογή Συν-προγραμματισμός με Socket**

**Ιχνηλασία**

****

**ΑΦΗΡΗΜΕΝΗ**

Οι αναδυόμενες εφαρμογές και τα δίκτυα προβάλλουν περισσότερες απαιτήσεις για αμφίδρομη ευαισθητοποίηση μεταξύ εφαρμογών και δικτύων. Έχει μελετηθεί πολύ τα τελευταία χρόνια πώς να ενοποιηθούν εφαρμογές και δίκτυα. Σε αυτό το άρθρο, παρατηρούμε ότι η υποδοχή είναι η μέση μεταξύ εφαρμογών και δικτύων, επομένως διερευνούμε τη δυνατότητα ενοποίησης εφαρμογών και δικτύων στο socket layer για πρώτη φορά. Εφαρμόζουμε για πρώτη φορά πλαίσιο ενσωμάτωσης δικτύου-εφαρμογών σε επίπεδο υποδοχής Socker με βάση το eBPF. Συσχετίζοντας υποδοχές με λειτουργίες ελέγχου δικτύου, οι προγραμματιστές μπορούν να πραγματοποιήσουν ευέλικτο έλεγχο δρομολόγησης με βάση τη λογική εφαρμογής, καθώς και δυναμική προσαρμογή λογικής εφαρμογής με βάση τις καταστάσεις του δικτύου. Στο πλαίσιο της προκαταρκτικής μας αξιολόγησης, το αποτέλεσμα δείχνει ότι η εφαρμογή που βασίζεται στο Socker επιτυγχάνει κατά μέσο όρο 28,5% μείωση χρόνου αιτήματος σε σύγκριση με την παραδοσιακή πρίζα υλοποίησης.

**ΕΝΝΟΙΕΣ CCS**

• Δίκτυα → Διεπαφές προγραμματισμού.

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**

Δίκτυα καθορισμένα από λογισμικό. Πρίζα; Ενσωμάτωση Δικτύου-Εφαρμογών

**1. ΕΙΣΑΓΩΓΗ**

Οι σημερινές κατανεμημένες εφαρμογές γίνονται όλο και πιο προσαρμοσμένες. Για να είναι πιο ανταγωνιστικοί στην αγορά εφαρμογών, απαιτούν υψηλή απόδοση με διάφορες μετρήσεις απόδοσης. Παρόμοια με τις εφαρμογές που εκτελούνται σε αυτόνομες μηχανές, οι οποίες βελτιστοποιούνται μόνες τους από τη χρήση των λεπτομερειών του υποκείμενου λειτουργικού συστήματος, είναι μια λογική αναλογία ότι οι κατανεμημένες εφαρμογές θα μπορούσαν επίσης να βελτιώσουν την απόδοση από τη γνώση της λογικής ελέγχου στο υποκείμενο δίκτυο. Από την άλλη, γίνονται και τα τρέχοντα δίκτυα όλο και πιο «ευφυής». Παράδειγμα SDN και δίκτυο με δυνατότητα NFV διαχείριση και ενορχήστρωση [2] δίνουν τη δυνατότητα στους χειριστές δικτύων να καθορίζουν πολύπλοκες πολιτικές για τη λειτουργία των δικτύων με ευέλικτο και δυναμικό τρόπο. Σε συνδυασμό με την αυξημένη δυνατότητα προγραμματισμού στο δίκτυο, φαίνεται επίσης δυνατό να ενσωματωθούν συγκεκριμένες απαιτήσεις εφαρμογής στη λογική ελέγχου δικτύου για την παροχή διαφοροποιημένων υπηρεσιών στις εφαρμογές.

Ως εκ τούτου, η σύζευξη των εφαρμογών και του ελέγχου του δικτύου έχει γίνει δημοφιλής τα τελευταία χρόνια. Από τη μία πλευρά, έχει καταβληθεί μεγάλη προσπάθεια για την ενημέρωση του δικτύου σχετικά με τις απαιτήσεις εφαρμογής. Για παράδειγμα, στα δίκτυα SDN της Google [4, 11] έχουν ήδη εφαρμοστεί λειτουργίες ελέγχου για τη συσσώρευση εφαρμογής δεδομένων και λαμβάνουν αποφάσεις ΤΕ για τα δεδομένα. Από την άλλη πλευρά, σίγουρα οι εφαρμογές έχουν επίσης τεχνικές προσαρμογής στην κατάσταση του δικτύου, οι οποίες επηρεάζονται από τον έλεγχο του δικτύου. Βασισμένο σε DASH(Διαδικασία επιλογής) οι εφαρμογές [10] προσαρμόζουν τους ρυθμούς bit με βάση το τρέχον εύρος ζώνης.

Και στο cloud gaming [12], βασίζονται οι αλγόριθμοι ανίχνευσης απόδοσης επένδυσης (ROI) μονόπλευρη καθυστέρηση. Επιπλέον, υπάρχουν επίσης μελέτες που συνδυάζουν εφαρμογές και έλεγχο δικτύου σε κλειστό βρόχο. Για παράδειγμα στη Minerva [8], ο αλγόριθμος ελέγχου συμφόρησης βασίζεται στον χρήστη QoE, και η επιλογή του ποσοστού από την άλλη πλευρά είναι σύμφωνα με το δίκτυο εύρος ζώνης.

Ωστόσο, όλες οι παραπάνω εργασίες προϋποθέτουν τη βασική λειτουργία της εφαρμογής η λογική (αναφέρεται ως AF) και η λογική της λειτουργίας ελέγχου δικτύου (αναφέρεται ως NF) βρίσκονται χωριστά σε διαφορετικά περιβάλλοντα. Οι εσωτερικές λεπτομέρειες του AF και του NF είναι σιωπηρές μεταξύ τους. Για να έχουν στενή συνεργασία, πρέπει επιπλέον να συνοψίσουν τις πληροφορίες τους σε σκίτσα και να ανταλλάξουν τα σκίτσα μεταξύ τους. Αυτό θα φέρει ένα οδυνηρό δίλημμα καθώς μικρά σκίτσα οδηγούν σε απώλεια πληροφόρησης και η συνεργασία θα είναι λιγότερο αποτελεσματική, ενώ μεγάλα σκίτσα θα μπορούσαν να προκαλέσουν μεγάλα έξοδα για την κατασκευή και το πέρασμα των σκίτσων, καθώς και να εκθέτουν πάρα πολλές πληροφορίες που μπορεί να εγείρει ζητήματα ασφάλειας και για τα δύο μέρη. Εξάλλου, όλα λειτουργούν απλώς για έναν συγκεκριμένο τύπο εφαρμογών ή σε συγκεκριμένο τύπο ελέγχου δικτύου. Δεν υπάρχει ένα γενικό πλαίσιο για κοινό προγραμματισμό των εφαρμογών και του ελέγχου δικτύου.

Προτείνουμε μια διαφορετική ιδέα από όλες αυτές στις προηγούμενες εργασίες: από κοινού εντοπίστε AF και NF που ελέγχει τις αντίστοιχες ροές AF κάτω από το ίδιο περιβάλλον κεντρικού υπολογιστή. Με αυτόν τον τρόπο, οι εφαρμογές και ο έλεγχος του δικτύου μπορεί να ανταλλάσσει απευθείας πληροφορίες χωρίς απώλεια πληροφοριών, οπότε η συνολική απόδοση θα βελτιωθεί σημαντικά. Επίσης, από όταν θα βρίσκονται στο ίδιο πλαίσιο , στους τερματικούς κεντρικούς υπολογιστές ,η ανησυχία για την ασφάλεια ,ότι δηλαδή οι εφαρμογές θα μπορούσαν να διαρρεύσουν σημαντικές πληροφορίες προς τα έξω δεν υπάρχει πια. Το NF θα μπορούσε να είναι το ενδιάμεσο σημείο για το αντίστοιχο AF για να επικοινωνεί τις πολιτικές δικτύου με το εκτός δίκτυο, διατηρώντας παράλληλα τις λεπτομέρειες εφαρμογής του AF μέσα. Τέλος, αυτή η πρόταση απεικονίζει ένα ενιαίο πλαίσιο τόσο για την AF όσο και για NF μέσα στο κατανεμημένο σύστημα. Στο μέλλον όταν μια υπηρεσία γίνεται δημοφιλής, το δίκτυο θα βοηθά τους χρήστες εφαρμογών να βελτιωθούν να κατανοούν και να διαχειρίζονται ολόκληρο το σύστημα.

Όσο πολλά υποσχόμενο ακούγεται, εξακολουθούν να υπάρχουν προκλήσεις για τη δέσμευση της AF και NF στο ίδιο πλαίσιο τελικού κεντρικού υπολογιστή:

**Πολύπλοκη εξάρτηση:** Οι χωριστοί κωδικοί AF και NF μετατρέπονται σε ένα στενά συνδεδεμένο περιβάλλον. Αλλά όλα τα περίπλοκα οι κλάδοι και οι συνθήκες στους αρχικούς κωδικούς καθιστούν δύσκολο τον εντοπισμό όλων των σημείων όπου η AF και η NF έχουν αλληλεπιδράσεις. Μπορεί να δημιουργηθεί AF με πολλαπλές ροές σε διαφορετικούς χρόνους κάτω από διαφορετικές καταστάσεις, και το NF μπορεί να έχει πολλούς κανόνες που μπορεί να ενεργοποιηθούν σε διαφορετικές εκδηλώσεις. Η πολυπλοκότητα στη στατική ανάλυση των κανόνων NF είναι υπεύθυνη για τις οποίες οι ροές AF είναι υψηλές.

**Γρήγορος συγχρονισμός:** Παλαιότερα, τα περισσότερα AF και NF εκτελούνταν ανεξάρτητα στο δικό τους περιβάλλον και περιοδικά αντλούσαν πληροφορίες με σκίτσα η μία από την άλλη. Όταν υπάρχουν απότομες αλλαγές που συμβαίνουν στη μέση δύο έλξεων, δεν μπορούν να αντιδράσουν για να συγχρονιστούν στις καταστάσεις του ενός στο χρόνο. Παρατηρούμε ότι η AF και η NF αλληλοεπιδρούν και οι δύο με τις υποδοχές. Η κλήση AF στις υποδοχές είναι για αποστολή/λήψη ροών ενώ το NF ρυθμίζει τις παραμέτρους υποδοχής για να εκφράσει τη λογική ελέγχου. Επομένως, προτείνουμε το Socker, μια πρίζα για την επίλυση αυτών των προκλήσεων. Για την επίλυση σύνθετων εξαρτίσεων AF και NF, χρησιμοποιείται δυναμική παρακολούθηση ροής ανίχνευση στις υποκείμενες υποδοχές και δυναμική σύνδεση τις AF και NF. Για να γίνει γρήγορος συγχρονισμός μεταξύ AF και NF, η αντίδραση είναι ανεξάρτητη από το πρωτόκολλο και χρησιμοποιείται για γρήγορη απόκριση σε αλλαγές κατά το χρόνο εκτέλεσης. Στο πλαίσιο της προκαταρκτικής μας αξιολόγησης, το αποτέλεσμα δείχνει ότι η εφαρμογή που βασίζεται στο Socker επιτυγχάνει κατά μέσο όρο μείωση του χρόνου αιτήματος κατά 28,5% σε σύγκριση με την παραδοσιακή εφαρμογή που βασίζεται σε πρίζα.

Συνοπτικά, κάνουμε τις ακόλουθες συνεισφορές:

• Παρακινούμε το σχέδιό μας και εξηγούμε γιατί είναι καλύτερο σε σύγκριση με τις λύσεις Strawman. (§2).

• Προτείνουμε το σχέδιο του Socker. (§3).

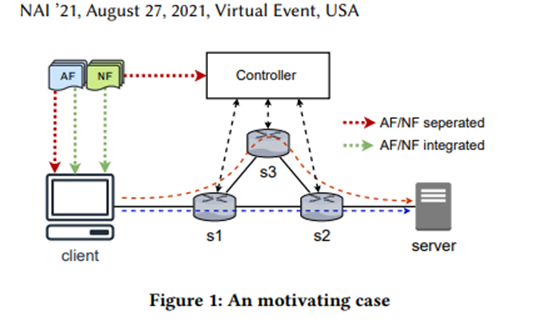
• Υλοποιούμε ένα πρωτότυπο Socker και διενεργούμε προκαταρκτική αξιολόγηση. (§4, §5).

• Συνοψίζουμε την προηγούμενη εργασία για τη σύζευξη της εφαρμογής και δίκτυο (§6).

**2 ΚΙΝΗΤΡΟ**

**2.1 Ενθαρρυντική περίπτωση**

Ξεκινάμε με μια απλή ρύθμιση για να δείξουμε ότι είναι απαραίτητο να εντοπιστεί AF και NF μαζί. Ο πελάτης εκτελεί μια εφαρμογή όπου ανεβάζει έναν ορισμένο αριθμό αρχείων στον διακομιστή. Συνδέονται με ένα μικρό δίκτυο με 3 κόμβους μεταγωγής, όπως στο Σχήμα 1. Η διαδρομή s1-s2 έχει μικρότερο λανθάνοντα χρόνο αλλά και μικρότερο εύρος ζώνης, ενώ αντ' αυτού είναι το s1-s3-s2 το αντίθετο. Το AF εδώ πρέπει να επιλέξει ένα επίπεδο συμπίεσης για συμπίεση στα αρχεία. Όσο υψηλότερο είναι το επίπεδο, τόσο μικρότερα θα είναι τα μεταδιδόμενα αρχεία, και φυσικά, τόσο μεγαλύτερος θα είναι ο χρόνος συμπίεσης. Ο NF εδώ πρέπει να επιλέξει μια διαδρομή από τον πελάτη στον διακομιστή για ανέβασμα του αρχείου. Ο στόχος είναι να ολοκληρωθεί η μεταφόρτωση το συντομότερο δυνατό.



Ο συνολικός χρόνος μεταφόρτωσης αποτελείται από το χρόνο συμπίεσης και το χρόνο μετάδοσης. Χαμηλότερο επίπεδο συμπίεσης σημαίνει μικρότερος χρόνος συμπίεσης, αλλά και μεγαλύτερο μέγεθος αρχείου στη μετάδοση. Εκτός, από το εύρος ζώνης της διαδρομής δικτύου και την καθυστέρηση που επηρεάζουν επίσης τον χρόνο μετάδοσης. Επομένως, η AF και η NF θα πρέπει να συνεργαστούν με σύνεση για να αποφασίσουν το επίπεδο συμπίεσης και διαδρομή με κοινό τρόπο.

Όταν το AF και το NF είναι ξεχωριστά, όπως φαίνεται με τα κόκκινα βέλη μέσα στο Σχήμα 1, ενδέχεται να μην είναι σε θέση να ανταλλάξουν πληροφορίες αποτελεσματικά. Σε αυτήν την περίπτωση, ιδανικά χωρίς συμφόρηση, το AF θα επιλέξει χαμηλό επίπεδο συμπίεσης και το NF θα επιλέξει το s1-s2 για να επιταχύνει τη μεταφόρτωση. Αλλά εάν το AF αρχίσει ξαφνικά να στέλνει ένα μεγάλο αρχείο χωρίς να ειδοποιήσει τον NF να αλλάξει τη διαδρομή εγκαίρως, το s1-s2 σύντομα θα υπερφορτωθεί. Από την άλλη, εάν το s1-s3-s2 είναι η εφεδρική διαδρομή για πολλαπλές AF, και όλα τα μικρά μονοπάτια δέχονται ξαφνική συμφόρηση, τότε το NF θα τα επαναδρομολογήσει όλα μέσω s1-s3-s2. Χωρίς ειδοποίηση από την AF για αύξηση του επιπέδου συμπίεσης και μείωση των ρυθμών αποστολής, το s1-s3-s2 θα λάβει σύντομα τεράστιο συσσωρευμένο όγκο ροών και επίσης θα συμφορηθεί . Για να λυθεί ένα τέτοιο χάος, οι χρήστες θα πρέπει να γράψουν ένα πρόσθετο πρωτόκολλο για ανταλλαγή πληροφορίας μεταξύ AF και NF και να ξαναγράψουν τους κώδικές τους με την προσθήκη του πρόσθετου πρωτόκολλου, το οποίο είναι επίπονο.

Αντίθετα, αν ενσωματώσουμε AF και NF στο ίδιο πλαίσιο στους τερματικούς κεντρικούς υπολογιστές, όπως φαίνεται με τα πράσινα βέλη στην Εικόνα 1. Συγκεκριμένα, τα προγράμματα ελέγχου δικτύου (NF) μετακινούνται στους τερματικούς κεντρικούς υπολογιστές, το AF και το NF μπορούν να ανταποκριθούν σε οποιεσδήποτε αλλαγές κατάστασης το ένα ή του άλλου γρήγορα και με αντιδραστικό τρόπο, έτσι ένας τελικός κεντρικός υπολογιστής γίνεται "ελεγκτής", αλλά μπορεί να ελέγχει μόνο τη ροή που προέρχεται από τον εαυτό της. Όπως φαίνεται στο Σχήμα 2, το μόνο που χρειάζεται να κάνουμε είναι απλώς να προσθέσουμε μερικούς σχολιασμούς με το shareable(κοινή χρήση) των μεταβλητών σε NF και AF. Το AF καθορίζει το επίπεδο συμπίεσης του αρχείου με βάση τις καταστάσεις δικτύου και το NF υπολογίζει τη διαδρομή δρομολόγησης (maxBWPath ή shortestPath) με βάση τον τύπο αρχείου. Αυτό είναι απλό στη χρήση χωρίς τροποποίηση των αρχικών κωδικών και προσθήκη επιπλέον πρωτοκόλλου επικοινωνίας.

**2.2 Λύσεις Strawman**

Στη συνέχεια συζητάμε την επιλογή σχεδίασης κατά τη σύνδεση AF και NF. Όπως αναφέρθηκε στην ενότητα 1, για τη διαχείριση σύνθετων εξαρτήσεων και του γρήγορου συγχρονισμού AF και NF, η βασική πρόκληση είναι να χειρίζονται αποτελεσματικά την επικοινωνία. Παρόμοια με την επικοινωνία μεταξύ των διεργασιών, η AF και η NF θα πρέπει να ανταλλάσσουν πληροφορίες μέσω ορισμένων κοινόχρηστων αντικείμενων. Για λύσεις Strawman, έχουμε:

**Υλοποίηση χώρου χρήστη:** κοινόχρηστα αντικείμενα μπορούν να δημιουργηθούν ως καθολικές μεταβλητές στο χώρο χρήστη. Αυτή η μέθοδος είναι εύκολο να εφαρμοστεί και ο χώρος χρήστη είναι ευέλικτος στην υποστήριξη. Ωστόσο, η AF δεν έχει ορατότητα στο χώρο του πυρήνα, επομένως δεν γνωρίζει το υποκείμενο λεπτομέρειες δικτύου, δηλαδή χαρακτηριστικό υποδοχής και κατάσταση, μιας ροής που παρήγαγε. Στη συνέχεια, η NF δεν μπορεί να χειριστεί μοναδικά αυτήν τη ροή ως δίκτυό της Η δυνατότητα δεν γράφεται από την αυτόματη εστίαση στα κοινόχρηστα αντικείμενα.

**Υλοποίηση χώρου πυρήνα:** Το eBPF [3] προσφέρει ένα ιδιαίτερο είδος της δομής δεδομένων που ονομάζονται χάρτες eBPF, οι οποίοι είναι παγκοσμίως προσβάσιμοι μεταξύ του χώρου του πυρήνα και του χώρου χρήστη. Δυνάμει των χαρτών eBPF, εμείς μπορούμε να συσχετίσουμε το AF με την υποδοχή που καλεί κατά την εκτέλεση και να αποθηκεύσει πληροφορίες σε χάρτες eBPF, ώστε το NF να μπορεί να ελέγχει και να ελέγχει μοναδικά με την αντίστοιχη πρίζα. Ωστόσο, οι χάρτες eBPF ουσιαστικά καταλαμβάνουν μια περιορισμένη περιοχή της μνήμης του πυρήνα και οι τύποι χαρτών είναι περιορισμένοι ανά πυρήνα, επιπλέον, οι χάρτες δεν υποστηρίζουν pub/sub API για να εκτελέσει αυτόματη επανεκτέλεση. Άρα αυτή η λύση δεν είναι κατάλληλη για το σύστημά μας.

Ο σχεδιασμός μας συνδυάζει τις δύο λύσεις Strawman. Για την AF, τα κοινόχρηστα αντικείμενα στο χώρο χρήστη θα μπορούσαν να αποθηκεύσουν πληροφορίες σε επίπεδο εφαρμογής και οι χάρτες eBPF θα μπορούσαν να αποθηκεύσουν πληροφορίες σε επίπεδο δικτύου, οι οποίες είναι σχετικά μικρές καθώς μόνο η ταιριαστή σχέση ζεύγους κλειδιού-τιμής εφαρμογής και υποδοχής χρειάζεται καταγραφή. Εξηγούμε λεπτομερώς αυτό το σχέδιο στην επόμενη ενότητα.

**3 ΣΧΕΔΙΑΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ**

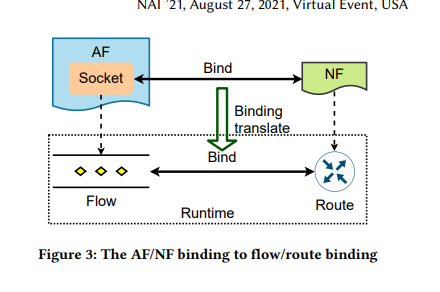
Σε αυτή την ενότητα, παρουσιάζουμε πρώτα τις παρατηρήσεις μας που οδηγούν στο σχεδιασμό, στη συνέχεια προτείνουμε το σχέδιό μας για το Socker και βουτάμε στη λεπτομέρεια περιγράφοντας την περίπτωση της σύνδεσης TCP στο Socker.

**3.1 Παρατηρήσεις σχεδιασμού**

Οι προγραμματιστές γράφουν AF για τη δημιουργία υποδοχών και NF για τον έλεγχο δικτύου. Μια υποδοχή μπορεί να συνδέσει ένα NF για να καθορίσει το επιθυμητό στοιχείο ελέγχου τα δεδομένα που είναι γραμμένα στην υποδοχή (δείτε παράδειγμα στο Σχήμα 2).



Ωστόσο, η τελική ροή που δημιουργείται από μια πρίζα μπορεί να προσδιοριστεί μόνο στο runtime (π.χ., οι παράμετροι socket() είναι μεταβλητές ή εφήμερες κατανομή λιμένων). Εν τω μεταξύ, το αποτέλεσμα εκτέλεσης (π.χ. μια διαδρομή) του NF είναι επίσης άγνωστο πριν από την εκτέλεση. Στην αφαίρεση υψηλού επιπέδου, ο Socker πραγματοποιεί τη δεσμευτική μετάφραση από το δέσιμο AF/NF κατά τον προγραμματισμό στη σύνδεση ροής/διαδρομής κατά το χρόνο εκτέλεσης, όπως φαίνεται στο Εικόνα 3. Για το σκοπό αυτό, έχουμε τις ακόλουθες παρατηρήσεις να οδηγήσουμε το σχέδιο του Socker.

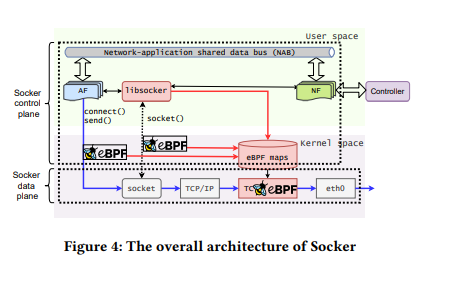


**Η επικοινωνία μεταξύ AF και NF θα πρέπει να είναι ανεξάρτητη από το πρωτόκολλο**. Όταν σχεδιάζουμε τους μηχανισμούς επικοινωνίας του AF και του NF, ένας φυσικός τρόπος που σκεφτήκαμε είναι με άμεση κλήση συνάρτησης μεταξύ τους. Ωστόσο, αυτό οδηγεί σε μια ισχυρή σύζευξη μεταξύ της AF και NF στην προοπτική του πρωτοκόλλου μεταξύ AF και NF, οποιαδήποτε αλλαγή με την παράμετρο οδηγεί σε πολλές αλλαγές των παρακλητών. Επίσης, είναι δύσκολο να πραγματοποιηθεί η αυτόματη επανεκτέλεση AF και NF όταν μια κατάσταση από τις μεταβλητές μέσα στο AF/NF άλλαξε. Έτσι, υιοθετούμε το παρατηρήσιμο μοτίβο σχεδίασης μεταξύ AF και NF, τα κοινόχρηστα αντικείμενα να αποθηκεύονται σε ένα αυτόνομο δίαυλο δεδομένων σε επίπεδο προγράμματος εξοπλισμένο με pub/sub API.

Η ροή που δημιουργείται από μια υποδοχή πρέπει να παρακολουθείται στον πυρήνα κατά το χρόνο εκτέλεσης για να αποκτήσετε την εφήμερη θύρα. Σκεφτείτε τη δημιουργία μιας ροής TCP, πριν από την κλήση του συστήματος connect(), η τοπική θύρα της πρίζας δεν έχει καθοριστεί. Ωστόσο, στο σύστημα connect() η κλήση οδηγεί σε τριμερή χειραψία TCP στον πυρήνα, άρα ένα σύνολοτων πακέτων έχουν ήδη αποσταλεί μετά την επιστροφή της κλήσης συστήματος connect(),οδηγώντας στο εκτός ελέγχου των πακέτων χειραψίας. Πρέπει λοιπόνναπαρακολουθούμε τη ροή μιας υποδοχής στο χώρο του πυρήνα, για αυτό το σκοπό, επιλέγουμεναχρησιμοποιήσουμε το σύνολο των εργαλείων eBPF για να εντοπίσουμε τις κλήσεις του συστήματος στον πυρήνα.

**3.2 Συνολική Αρχιτεκτονική**

Η συνολική αρχιτεκτονική του Socker αποτελείται από το επίπεδο ελέγχου και το επίπεδο δεδομένων όπως φαίνεται στο Σχήμα 4.



Τα κόκκινα βέλη υποδεικνύουν το διαδρομές ελέγχου και τα μπλε βέλη είναι οι διαδρομές δεδομένων.

**Αεροπλάνο ελέγχου κάλτσας**. Το επίπεδο ελέγχου του Socker είναι υπεύθυνο για τρεις εργασίες.

(1) Παροχή κοινόχρηστου επιπέδου δεδομένων για AF και NF και έναυσμα για επανεκτέλεση αυτόματα.

(2) Παρακολούθηση των πληροφοριών περιβάλλοντος (δηλαδή, ο χώρος χρήστη πρίζες) της εφαρμογής.

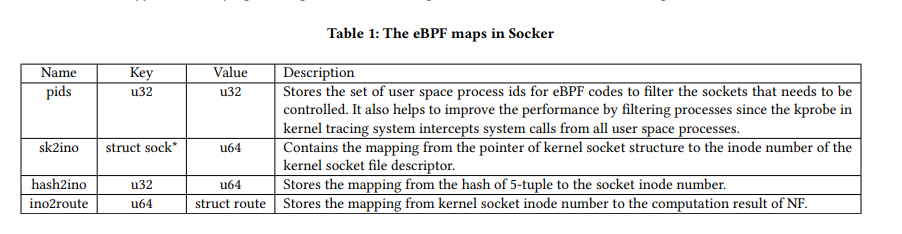
(3) Συγχρονισμός των επιθυμητών καταστάσεων δικτύου (π.χ. η επιθυμητή διαδρομή για μια ροή) στο επίπεδο δεδομένων

Αυτά επιτυγχάνονται με έναν κοινόχρηστο δίαυλο δεδομένων εφαρμογής δικτύου (NAB), ένα libsocker βιβλιοθήκης υψηλού επιπέδου που εξαρτάται από τη γλώσσα και ένα σύνολο προγραμμάτων eBPF. Για να επιτευχθεί η αλληλεπίδραση μεταξύ της AF και NF πραγματοποιούν αυτόματη επανεκτέλεση όταν αλλάζει η κατάσταση. Σχεδιάζουμε το NAB ως δίαυλο δεδομένων, έτσι το AF και το NF μπορούν να μοιραστούν το δικό τους που δηλώνετε με τρόπο ανεξάρτητο από πρωτόκολλο. Επίσης, το NAB μας υιοθετεί το παρατηρήσιμο μοτίβο σχεδίασης με pub/sub API που αυτόματα καταχωρούν το AF/NF ως παρατηρητές στα εξαρτώμενα κράτη τους, άρα οποιαδήποτε αλλαγές στις καταστάσεις μπορεί να πυροδοτήσουν την εκ νέου εκτέλεση του AF/NF. Οι καταστάσεις δικτύου αποσύρονται από το NF και από τον ελεγκτή και δημοσιεύονται στο NAB. Το libsocker τυλίγει την αρχική υποδοχή επιπέδου χρήστη λειτουργίας και ένα σύνολο προγραμμάτων eBPF που συνδέονται με κλήσεις συστήματος. Οι εφαρμογές χρησιμοποιούν το libsocker για τη δημιουργία υποδοχών, άρα τον χρόνο εκτέλεσης της διαδικασίας πληροφορίας (π.χ. αναγνωριστικό διεργασίας, περιγραφέας αρχείου υποδοχής) μπορούν να συλληφθούν από το Socker. Επίσης, τα προγράμματα eBPF παρακολουθούν τις λειτουργίες του πυρήνα (π.χ. sock\_alloc\_file(2)/tcp\_connect(2)) που σχετίζονται με τις υποδοχές που δημιουργούνται από εφαρμογές και αποθηκεύουμε τις καταστάσεις που παρακολουθούνται σε χάρτες eBPF του πυρήνα. Τα προγράμματα eBPF συνδέονται με κλήσεις συστήματος υποδοχής, παρακολουθούμε τα ορίσματα που μεταβιβάζονται και το σύστημα καλεί και αποθηκεύει τις πληροφορίες στους χάρτες eBPF.

Επιπλέον, ο συγχρονισμός μεταξύ του NF και της πολιτικής δικτύου απαιτεί ορισμένους μηχανισμούς που επιτρέπουν στους κεντρικούς υπολογιστές να κάνουν έγχυση διαμορφώσης δικτύου στη διαδρομή δεδομένων. Για παράδειγμα, το NF επικοινωνεί με τον ελεγκτή δικτύου χρησιμοποιώντας ένα αποκλειστικό κανάλι. Δεδομένου ότι η τεχνική δρομολόγησης πηγής μπορεί να επιτύχει καθορισμένο τελικό κεντρικό υπολογιστή δρομολόγησης χωρίς την εμπλοκή του ελεγκτή, η οποία μπορεί σε μεγάλο βαθμό να απλοποιήσει τη ρύθμισή μας, επομένως υποθέτουμε ότι το δίκτυο υποστηρίζει την πηγή δρομολόγησης, και περιγράφουμε την αρχιτεκτονική του Socker με βάση τη ρύθμιση δρομολόγησης πηγής.

**Socker Data Plane.** Το επίπεδο δεδομένων Socker είναι υπεύθυνο για να εφαρμόσει τις επιθυμητές ενέργειες που επιστρέφονται από το NF στα εξερχόμενα πακέτα, αυτό γίνεται με την προσάρτηση ενός προγράμματος eBPF στην κυκλοφορία εξόδου έλεγχος (TC). Για παράδειγμα, σε ένα δίκτυο δρομολόγησης πηγής, εάν το NF είναι μια συνάρτηση ελέγχου δρομολόγησης και επιστρέφει μια διαδρομή για τη ροή, το eBPF θα ενθυλακώσει τα πακέτα της ροής πριν από την εκπομπή στις διεπαφές δικτύου.

**Χάρτες Socker eBPF.** Ο κύριος στόχος των χαρτών eBPF στο Socker είναι η αποθήκευση της αντιστοίχισης από τη ροή που δημιουργείται από τις υποδοχές χώρου χρήστη στα αποτελέσματα εκτέλεσης του NF. Ωστόσο, δεν υπάρχει ξεκάθαρος τρόπος σύνδεσής τους (π.χ., ο χώρος χρήστη δεν μπορεί να πάρει απευθείας τον πυρήνα διεύθυνση δομής υποδοχής). Για το σκοπό αυτό, σχεδιάζουμε τέσσερους βασικούς eBPF χάρτες τύπου BPF\_MAP\_TYPE\_HASH όπως φαίνεται στον Πίνακα 1 και χρήση ο αριθμός inode της δομής υποδοχής πυρήνα ως γέφυρα για σύνδεση στις υποδοχές χώρου χρήστη και την υποδοχή χώρου πυρήνα, επειδή ο χώρος χρήστη στο πρόγραμμα μπορεί να διαβάσει το σύνδεσμο αρχείου στο σύστημα αρχείων Linux /proc και να πάρει τον αριθμό inode της αντίστοιχης υποδοχής πυρήνα inode.



**Προγράμματα Socker eBPF.** Το Socker περιέχει ένα σύνολο προγραμμάτων eBPF που συνδέονται με κλήσεις συστήματος για να παρακολουθείτε τις πρίζες και τις συνδέσεις TCP/UDP που δημιουργούνται από εφαρμογές, επιπλέον, επισυνάπτουμε ένα πρόγραμμα eBPF

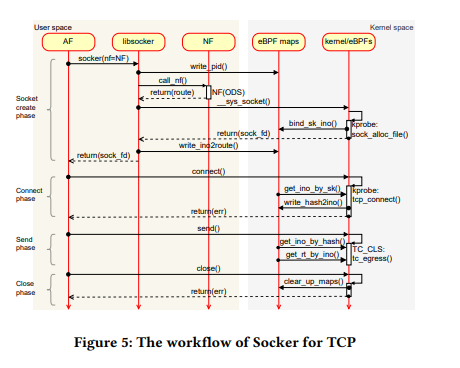
• **kprobe:sock\_alloc\_file.** Σε μια πρίζα δημιουργήστε κλήση συστήματος , όταν κληθεί ο πυρήνας θα κατασκευάσει μια δομή υποδοχής πυρήνα που σχετίζεται με τη δομή του αρχείου. Συνδέουμε ένα kprobe eBPF συνάρτηση στο αρχείο sock\_alloc για να λάβετε την αντιστοίχιση από δομή υποδοχής πυρήνα και τον αριθμό του κωδικού και αποθηκεύστε το αντιστοίχιση στον χάρτη sk2ino. Όταν καλεί το σύστημα υποδοχής(2) επιστρέφει στο πρόγραμμα χρήστη, το libsocker ρωτά το inode αριθμός από τον κατάλογο του συστήματος αρχείων (/proc/pid/fs/fd) για να λάβετε τον αριθμό inode της υποδοχής πυρήνα και, στη συνέχεια, εκτελεί το NF για λήψη την επιθυμητή διαδρομή και αποθηκεύει την αντιστοίχιση από inode σε διαδρομή στον χάρτη ino2route.

**• kprobe:tcp\_connect.** Επισυνάπτουμε ένα πρόγραμμα kprobe eBPF στη συνάρτηση tcp\_connect στον πυρήνα. Δημιουργούμε τον γάντζο εδώ επειδή όταν καλείται η συνάρτηση tcp\_connect, η τοπική θύρα της υποδοχής έχει ήδη καθοριστεί, αλλά το πακέτο TCP SYN δεν εκπέμπεται. Αυτό μας κάνει να μπορούμε να ελέγχουμε τα πακέτα SYN. Υπολογίζουμε τον κατακερματισμό του 5-πλού και αναζητούμε το sk2ino λάβετε τον αριθμό εισόδου της τρέχουσας υποδοχής και, στη συνέχεια, αποθηκεύστε το αντιστοίχιση στον χάρτη hash2ino.

**• SCHED\_CLS:tc\_egress.** Επισυνάπτουμε το πρόγραμμα eBPF στην έξοδος του TC, το πρόγραμμα υπολογίζει τον κατακερματισμό του 5-tup από το \_\_sk\_buff και αναζητήστε το hash2ino για να λάβετε τον πυρήνα υποδοχή αριθμού inode και, στη συνέχεια, αναζητήστε τον χάρτη ino2route για να λάβετε την επιθυμητή διαδρομή για το τρέχον πακέτο. Πριν από την εκπομπή στο δίκτυο, το πακέτο θα υποβληθεί σε κατάλληλη επεξεργασία για να χωρέσει την υποδομή δικτύου (π.χ. ενθυλάκωση του πακέτου με Στοίβα MPLS για δίκτυα MPLS).

**3.3 Ροή εργασιών Socker**

Για να δείξουμε πώς ο σχεδιασμός του Socker πραγματοποιεί τον κοινό έλεγχο της εφαρμογής και του δικτύου, χρησιμοποιούμε τη σύνδεση TCP ροή εργασιών στο Socker ως περίπτωση για την περιγραφή της διαδικασίας όπως φαίνεται στην Εικόνα 5, όπου χωρίζουμε τη διάρκεια ζωής μιας υποδοχής TCP σε τέσσερα φάσεις. Για υποδοχές UDP που δεν καλούν το σύστημα connect() κλήση, εκτελούμε μια παρόμοια διαδικασία κλήσης συστήματος TCP connect() στην κλήση συστήματος send() της υποδοχής UDP.



**Φάση δημιουργίας υποδοχής.** Η εφαρμογή καλεί τη συνάρτηση socker() από το libsocker για να δημιουργήσει μια υποδοχή μαζί με μια λειτουργία δικτύου δείκτη ως μία από τις παραμέτρους. Το libsocker γράφει το ρεύμα επεξεργάζεται το αναγνωριστικό στον χάρτη eBPF του pids και καλεί το NF για να λάβει το αποτέλεσμα. Εν τω μεταξύ, το libsocker καλεί την κλήση του συστήματος υποδοχής, εμείς ανιχνεύουμε την εκτέλεση της συνάρτησης πυρήνα sock\_alloc\_file και αποθηκεύουμε την αντιστοίχιση από τη δομή της υποδοχής πυρήνα στον αριθμό inode που αντιπροσωπεύει την περιγραφή του αρχείου υποδοχής. Μετά την κλήση του συστήματος πρίζας επιστρέφει, το libsocker ρωτά τις πληροφορίες του συνδέσμου από το χώρο χρήστη περιγραφέας αρχείου υποδοχής για να λάβετε τον αριθμό inode της υποδοχής πυρήνα. Τέλος, το libsocker αποθηκεύει την αντιστοίχιση από τον αριθμό inode έως το αποτέλεσμα του NF στον χάρτη ino2route eBPF και το επιστρέφει στην εφαρμογή.

**Φάση σύνδεσης**. Όταν η εφαρμογή εκπέμπει ένα σύστημα σύνδεσης κλήση, παρακολουθούμε τη συνάρτηση του πυρήνα tcp\_connect() πριν από το SYN αποστέλλονται πακέτα. Το πρόγραμμα eBPF αναζητά τον χάρτη sk2ino eBPF για να λάβετε τον αριθμό inode της υποδοχής πυρήνα και να υπολογίσετε τον κατακερματισμό του 5- πλειάδες, στη συνέχεια αποθηκεύει την αντιστοίχιση από τον κατακερματισμό στον αριθμό inode στον χάρτη hash2ino eBPF. Σημειώστε ότι για τις υποδοχές UDP, τα παραπάνω η διαδικασία μεταφέρεται στη φάση αποστολής.

**Αποστολή φάσης.** Αφού δημιουργηθεί η σύνδεση TCP, η εφαρμογή στέλνει πακέτα στην υποδοχή μέσω κλήσεων συστήματος εγγραφής υποδοχής (π.χ. αποστολή/αποστολή()). το πρόγραμμα eBPF που συνδέεται με την έξοδο της κυκλοφορίας ο έλεγχος (TC) λαμβάνει το buffer υποδοχής πριν εκπέμψει το πακέτο σε διεπαφές δικτύου. Όταν ένα πακέτο φτάσει στην έξοδο TC, το πρόγραμμα eBPF υπολογίζει τον κατακερματισμό του 5-ταπλού από το buffer υποδοχής, και αναζητά τον χάρτη hash2ino για να πάρει τον αριθμό inode της υποδοχής πυρήνα και, στη συνέχεια, αναζητά το ino2route για να πάρει την επιθυμητή διαδρομή για το τρέχουντα πακέτο. Τέλος, ξαναγράψτε το πακέτο (π.χ. ενθυλάκωση με MPLS) πριν το εκπέμψετε στη διεπαφή δικτύου.

**Κλείσιμο φάσης.** Όταν η πρίζα είναι ρητά κλειστή από την εφαρμογή ή κλειστή λόγω σφαλμάτων, το Socker καθαρίζει όλες τις σχετικές καταχωρήσεις στους χάρτες eBPF.

**4 ΥΛΟΠΟΙΗΣΗ**

Υλοποιήσαμε ένα πρωτότυπο Socker βασισμένο στο BCC [3](v0.20.0). Για το libsocker, υλοποιήσαμε μια βιβλιοθήκη περιτυλίγματος υποδοχών Python. Το NAB υλοποιείται από μια τάξη python που ακολουθεί το παρατηρήσιμο μοτίβο σχεδίασης, εκθέτουμε το subscribe API με συνάρτηση python σχολιασμό, ο οποίος θα ενεργοποιήσει την περιτυλιγμένη συνάρτηση όταν τα εγγεγραμμένα δεδομένα άλλαξαν και τα δεδομένα μεταβιβάζονται στο περιτύλιγμα λειτουργούν ως παράμετροι (βλ. Εικόνα 2 ως παράδειγμα). Ο κωδικός είναι ήδη δημόσιος.

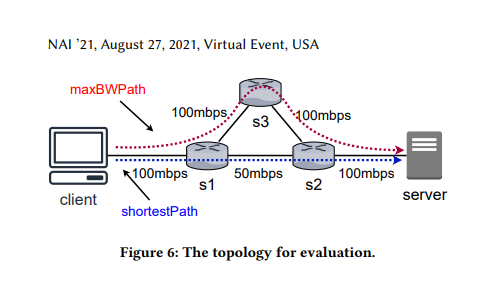
**5 ΑΞΙΟΛΟΓΗΣΗ**

Σε αυτή την ενότητα, δίνουμε μια προκαταρκτική αξιολόγηση του Socker σε ένα προσομοιωμένο δίκτυο, στόχος μας είναι να δείξουμε το όφελος κατά τη χρήση για να κάνει προγραμματισμό πρίζας ενάντια στον παραδοσιακό προγραμματισμό πρίζας.

**5.1 Ρυθμίσεις**

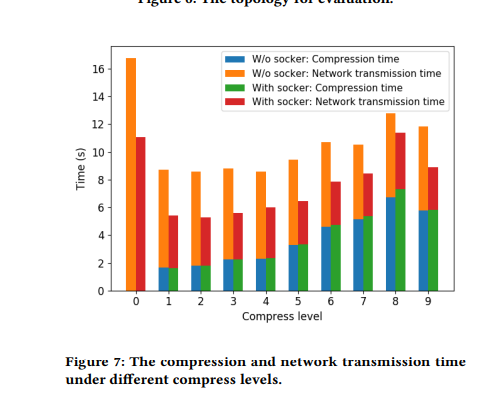
Η αξιολόγηση πραγματοποιείται σε επιφάνεια εργασίας Ubuntu20.04.1 (Linux πυρήνας 5.8.0 με ενεργοποιημένες μονάδες πυρήνα eBPF) με Intel i7 CPU (2,2 GHz) και μνήμη 16 GB. Κατασκευάσαμε ένα δίκτυο που αποτελείται από τρείς διακόπτες και δύο κεντρικούς υπολογιστές όπως φαίνεται στο Σχήμα 6 χρησιμοποιώντας mininet. Το δίκτυο έχει ρυθμιστεί ώστε να υποστηρίζει τη δρομολόγηση MPLS και το εύρος ζώνης που διατίθεται για κάθε σύνδεσμο σημειώνεται στο σχήμα. Ο διακομιστής εκτελείται σαν ένας απλός διακομιστής TCP για λήψη δεδομένων και ο πελάτης εκτελεί το πρόγραμμα που χρησιμοποιεί το Socker στο Σχήμα 2. Η αξιολόγηση βασίζεται σε μια στατική προβολή δικτύου, οι δυναμικές καταστάσεις βασίζονται σε μια οθόνη δικτύου (π.χ. στον ελεγκτή). Η έγκαιρη ενημέρωση από την οθόνη μπορεί να είναι στα NF των τελικών κεντρικών υπολογιστών, στη συνέχεια οι καταστάσεις διαδίδονται. Χρησιμοποιούμε το enwik8 (αρχείο κειμένου 100 MB) από τη Μεγάλη Συμπίεση κειμένου Σημείο αναφοράς [7] ως είσοδος του προγράμματος.

Γράψαμε ένα άλλο πρόγραμμα κανονικής υποδοχής για να κάνουμε την ίδια εργασία όπως στο σχήμα 2, τρέχουμε τα δύο προγράμματα με διαφορετική συμπίεση επίπεδα για να έχετε τη συνολική απόδοση.



**5.2 Αποτελέσματα**

Τα αποτελέσματα φαίνονται στο Σχήμα 7. Μπορούμε να δούμε ότι κάτω από διαφορετικά επίπεδα συμπίεσης, το Socker ξεπερνά πάντα το κανονικό πρόγραμμα υποδοχής και πέτυχε πάνω από 28,5% μείωση του χρόνου μεταφόρτωσης κατά μέσο όρο. Ο λόγος για το κέρδος μας είναι ότι το Socker επιτρέπει στην εφαρμογή να επιλέξει τις καλύτερες διαδρομές για τυπικές ροές, στην περίπτωσή μας, στις μεγάλες ροές μεταφόρτωσης αρχείων χρησιμοποιεί μια διαδρομή (δηλ. maxBWPath) με μεγαλύτερο εύρος ζώνης (λίγο μεγαλύτερη καθυστέρηση) σε σύγκριση με την προεπιλεγμένη διαδρομή (δηλ. συντομότερο μονοπάτι). Επίσης, η εφαρμογή μπορεί να ρυθμίσει το επίπεδο συμπίεσής της σύμφωνα με το εύρος ζώνης της διαδρομής και τις απαιτήσεις QoS που είναι το καθήκον της συνάρτησης computeCL στο σχήμα 2, στο σχεδιασμό του η συνάρτηση computeCL είναι εκτός του πεδίου εφαρμογής μας, ωστόσο, αξιολογούμε όλο το δυνατό επίπεδο συμπίεσης (από 1-9) του Gzip και το αποτέλεσμα δείχνει ότι το Socker μπορεί να κερδίσει σε όλες τις περιπτώσεις.



**6. ΣΧΕΤΙΚΗ ΕΡΓΑΣΙΑ**

Έλεγχος δικτύου με τη βοήθεια εφαρμογών: Υπάρχουν πολλές έρευνες που επιτρέπουν στον έλεγχο δικτύου να λειτουργεί με πληροφορίες εφαρμογών. Το Socket Intents [9] επαυξάνει τη διεπαφή υποδοχής σύμφωνα με τις απαιτήσεις της εφαρμογής. Το SDN WAN της Google [4] και το edge δίκτυο [11] χρησιμοποιούν τοπικούς ελεγκτές για τη συλλογή καταστάσεων εφαρμογής και τα παραδίδει στον κεντρικό ελεγκτή για να αποφασίσει την κατανομή της κυκλοφορίας. Για συγκεκριμένους τύπους εφαρμογών όπως το βίντεο, υπάρχουν επίσης μελέτες [1, 8, 13] για συνδυασμό μετρήσεων σε επίπεδο εφαρμογής, όπως το QoE σε οδηγός ελέγχου σε επίπεδο μεταφοράς.

**Εφαρμογή με εντολή δικτύου**: Οι εφαρμογές θα μπορούσαν επίσης να βελτιώσουν την απόδοση συνδυάζοντας την κατάσταση του δικτύου. Βασισμένο σε DASH οι αλγόριθμοι [10] προσαρμόζουν τον ρυθμό μετάδοσης bit με βάση την εκτίμηση του δικτύου το εύρος ζώνης, το οποίο υπολογίζεται από καταστάσεις δικτύου σε πραγματικό χρόνο. Σε τομείς της μηχανικής μάθησης και του cloud gaming [12], υπάρχουν επίσης εφαρμογές που επιλέγουν μεταξύ διαφορετικών αλγορίθμων με βάση τον λανθάνοντα χρόνο και το εύρος ζώνης του δικτύου. Ένα μειονέκτημα για αυτές τις εφαρμογές είναι ότι οι εφαρμογές μπορούσαν μόνο έμμεσα να «μαντέψουν» τον έλεγχο δικτύου από στατιστικές επιπέδου δεδομένων. Αυτό μπορεί να οδηγήσει σε ανακριβή και άκαιρα συμπέρασμα όταν η κατάσταση του δικτύου είναι ασταθής.

**Υπολογισμός εντός δικτύου:** Ένα άλλο ενδιαφέρον και σχετικό θέμα είναι η εκτέλεση εφαρμογών στο δίκτυο. Καθώς η δυνατότητα προγραμματισμού των κόμβων δικτύου αυξάνεται, οι εφαρμογές θα μπορούσαν να εκφορτώσουν μέρος της λογικής των κόμβων τους στο δίκτυο για να κάνει κάποιους υπολογισμούς κατά την πτήση. Με αυτόν τον τρόπο η μέση καθυστέρηση μετ' επιστροφής και ο υπολογισμός στους διακομιστές θα μπορούσαν και τα δύο να μειωθούν σε μεγάλο βαθμό. Οι τρέχουσες μελέτες [5, 6] αξιοποιούν προγραμματιζόμενους διακόπτες, και ως επί το πλείστων εφαρμόζουν απλές και αθροιστικές λειτουργίες και γρήγορο κλειδί-v.

Ο απώτερος στόχος του Socker είναι να παρέχει ένα ενιαίο πλαίσιο για όλους το AF και το NF, και να τα εφαρμόσουν σε ολόκληρη τη διανομή συστήματος συμπεριλαμβανομένων των κόμβων δικτύου. Το Socker θα μπορεί να παρέχει ένα μοντέλο προγραμματισμού υψηλού επιπέδου για εφαρμογές εντός δικτύου που αλληλοεπιδρούν με τον έλεγχο δικτύου.

**7. ΣΥΜΠΕΡΑΣΜΑ**

Σε αυτό το άρθρο, παρουσιάζουμε το Socker, ένα πρώτο πλαίσιο ενσωμάτωσης εφαρμογών δικτύου σε επίπεδο υποδοχής. Παρουσιάζουμε το σχεδιασμό του συστήματος του Socker αναλυτικά και δείχνουμε τη διαδικασία σύνδεσης TCP μέσο Κάλτσας. Τέλος, πραγματοποιούμε μια προκαταρκτική αξιολόγηση της υλοποίησης του πρωτοτύπου μας για να δείξουμε την επίδραση του Socker.

**REFERENCES**

[1] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2018. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). 267– 282.

[2] Kai Gao, Taishi Nojima, and Y Richard Yang. 2018. Trident: toward a unified sdn programming framework with automatic updates. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 386–401.

[3] iovisor. 2021. BCC: Tools for BPF-based Linux IO analysis, networking, monitoring, and more. https://github.com/iovisor/bcc. (2021).

[4] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Communication Review 43, 4 (2013), 3–14.

[5] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. Netchain: Scale-free sub-rtt coordination. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). 35–49.

[6] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. Netcache: Balancing key-value stores with fast in-network caching. In Proceedings of the 26th Symposium on Operating Systems Principles. 121–136.

[7] Matt Mahoney. 2021. Large Text Compression Benchmark. http://mattmahoney. net/dc/text.html. (2021).

[8] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. 2019. End-to-end transport for video QoE fairness. In Proceedings of the ACM Special Interest Group on Data Communication. 408–423.

[9] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. 295–300.

[10] Iraj Sodagar. 2011. The mpeg-dash standard for multimedia streaming over the internet. IEEE multimedia 18, 4 (2011), 62–67.

[11] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 432–445.

[12] Yunfei Zhang, Gang Li, Chunshan Xiong, Yixue Lei, Wei Huang, Yunbo Han, Anwar Walid, Y Richard Yang, and Zhi-Li Zhang. 2020. MoWIE: Toward Systematic, Adaptive Network Information Exposure as an Enabling Technique for Cloud-Based Applications over 5G and Beyond. In Proceedings of the Workshop on Network Application Integration/CoDesign. 20–27.

[13] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. 2019. Learning to coordinate video codec with transport protocol for mobile video telephony. In The 25th Annual International Conference on Mobile Computing and Networking. 1–16